Machine Learning – Preparing data

Michael Claudius, Associate Professor, Roskilde
Jens Peter Andersen, Assistant Professor, Roskilde

02.09.2020

Machine Learning Project

- ***Machine Learning has a number of phases***
- ***The phases can be overlapping and/or iterative***

  1. **Look at the big picture.**
  2. **Get the data.**
  3. **Discover and visualize the data to gain insights.**
  4. **Prepare the data for Machine Learning algorithms.**
  5. **Select a model and train it.**
  6. **Fine-tune your model.**
  7. **Present your solution.**
  8. **Launch, monitor, and maintain your system.**

- **A detailed checklist is given on** ML Management Checklist (PDF)

- **Remember always adapt the order and the checklist to your needs**

Machine Learning: The big picture and data

- **It is about understanding business and the data!**

1. **The context**
2. **Frame the problem**
3. **Select performance measure**
4. **Setup workspace**
5. **Get the data in hand**
6. **Explore the data tables**
7. **Create a test set**
8. **Visual graphs and correlations**
9. **Experiment with attribute combinations**

The Context: Housing prices

- **California median housing price for a block group**
- **Block group: unit population of 600-3.000 people**
- **Data size app. 20.000**

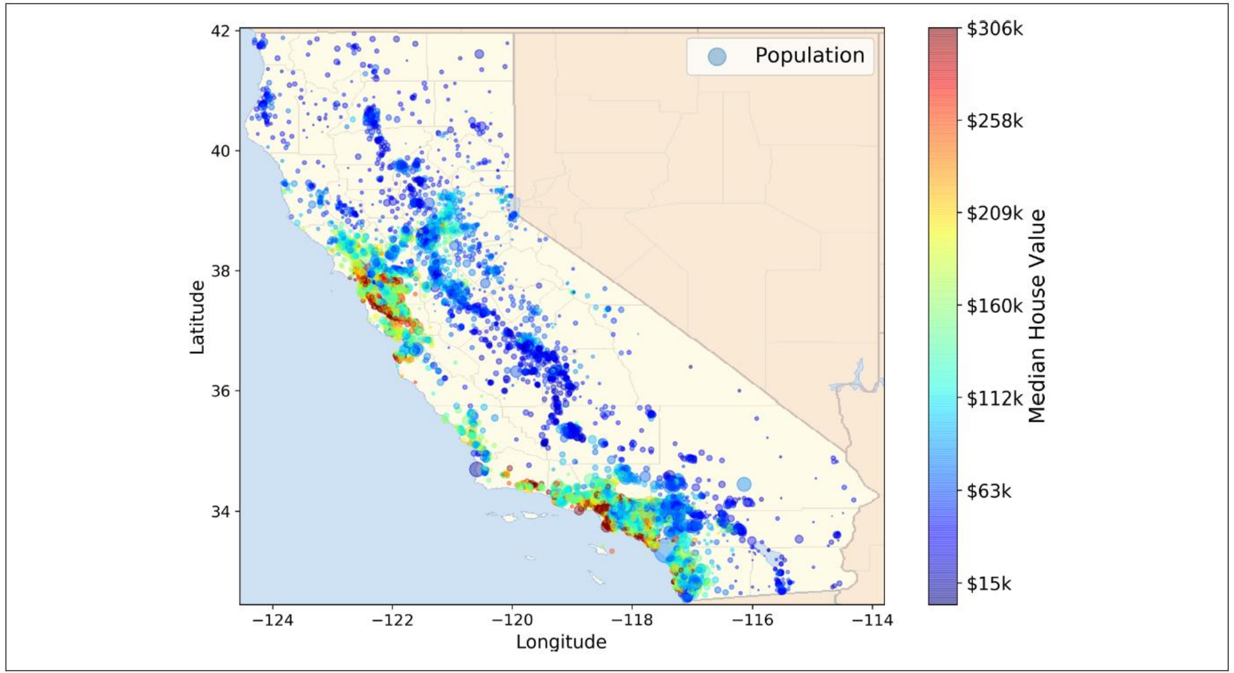

Figure 2-1. California housing prices

Prepare Data for Machine Learning Algorithms

Its about making data ready to be used by Machine Learning algorithms

1. **Data Cleaning**: Handle missing feature values -> How to fix it?
2. **Handle Text and Categorical Attributes**: Convert values to numbers so they can handled by ML algorithms
3. **Feature Scaling**: Scale features to the same order of magnitude: E,g intervals [-1..1] or [0..1] or distribution around median
4. **Custom Transformers**: Custom transformer classes on data can be defined to be used in e.g. transformations pipelines
5. **Transformation Pipelines**: Feature from Scikit-Learn, that automatically e.g. can apply the transformers

Create clean data sets

Revert to a clean training set and separate features and labels

```
housing = strat_train_set.drop("median_house_value", axis=1)
housing_labels = strat_train_set["median_house_value"].copy()
```

Data Cleaning

- **Problem:** Some feature values are missing – e.g. some registrations of "*total_bedrooms*" are missing.
  In general learning algorithms assume that feature values are <u>not</u> missing

Solutions:
1. Drop feature instances (rows) with missing values - e.g. those with missing values for "total_bedrooms"
2. Drop the whole attribute with missing values - e.g. the "total_bedrooms" feature
3. Replace the missing feature values with proper values – e.g. zero, mean or median values

```
housing.dropna(subset=["total_bedrooms"])    # option 1
housing.drop("total_bedrooms", axis=1)       # option 2
median = housing["total_bedrooms"].median()  # option 3
housing["total_bedrooms"].fillna(median, inplace=True)
```

- **How to apply strategy (median) on whole data set? Use SimpleImputer from sklearn.impute**

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
```

- **Check out the code (cell 55) in the Housing project !**

Data Cleaning

- **Problem:** Assume there are many features (>10). It is a tedious job manually handling missing values.

Solutions:

1. Use SimpleImputer from sklearn.impute
2. Apply a strategy (median) on whole data set?
3. Drop non-numerical values
4. Fit and transform the data set

```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy="median")
housing_num = housing.drop("ocean_proximity", axis=1)
imputer.fit(housing_num)
X = imputer.transform(housing_num)
```

Later one can add the transformed non-numerical values

Text and Categorical Attributes

Problem: Some features are discrete text/categorical attributes and must be converted to a (continuous) number equivalent.
Solution: Convert each category to a discrete number
Category: E.g. the “ocean_proximity” feature has values like: 'OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'

```
from sklearn.preprocessing import OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
housing_cat_encoded = ordinal_encoder.fit_transform(housing_cat)
housing_cat_encoded[:10]
array([[0.], [0.],[4.],[1.],[0.],[1.],[0.],[1.],[0.], [0.]])
```

Converted into five numbers: 0, 1, 2, 3, 4

Problem: ML-algorithm assumes nearby numbers have similarity. But 0 (‘OCEAN’) and 4(‘NEAR OCEAN’) have similarity.

Don’t worry there is a solution: Use *OneHotEncoder* from *sklearn.preprocessing* ☺

Text and Categorical Attributes, OneHotEncoder

Encoding: Each discrete feature is converted to a feature vector with a dimension equal number of values in range
Category: E.g. the "ocean_proximity" feature has values like: 'OCEAN', 'INLAND', 'ISLAND', 'NEAR BAY', 'NEAR OCEAN'
Converted into a table (matrix with 5 columns and 'one hot bit' in each row)

```
array([[1., 0., 0., 0., 0.],
  [1., 0., 0., 0., 0.],
  [0., 0., 0., 0., 1.],
……………....,
 [0., 1., 0., 0., 0.],
 [1., 0., 0., 0., 0.],
 [0., 0., 0., 1., 0.]])
```

Problem: Too many zeros in the matrix, Vasting memory!
Solution: Use OneHotEncoder with fit.transform creates utomatically a *sparse* matrix with position numbers and the value.
[Sparse Matrix explained]()

```python
from sklearn.preprocessing import OneHotEncoder
cat_encoder = OneHotEncoder()
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

Feature Scaling

Problem: Learning algorithms is assumed to perform better if numerical attributes in on the same scale
Solution: There are 2 common ways to get all attributes to have the same scale:

1. Min-max scaling:

Values are scaled to the range 0..1. NewValue = Value/(MaximumValue - MinimumValue).
Scikit-Learn provides the transformer **`MinMaxScaler`**.
Affected by outliers. E.G. one wrong house price (100) will place all other houses in a range 0-0.15

2. Standardization scaling:

Values scaled to a unit variance distribution with a mean around 0.
NewValue = (Value-Mean)/StandardDeviation. Typical interval [-3..3] holds 99% of values
Scikit-Learn provides the transformer **`StandardScaler`**
Less affected by outliers.

$$\sigma = \sqrt{\frac{\sum(x_i - \mu)^2}{N}}$$

σ = population standard deviation
N = the size of the population
x_i = each value from the population
μ = the population mean

Custom Transformers

Problem: Custom transformations on data may be needed – e.g. for providing calculated features:

```
housing["rooms_per_household"] = housing["total_rooms"]/housing["households"]
housing["bedrooms_per_room"] = housing["total_bedrooms"]/housing["total_rooms"]
housing["population_per_household"]=housing["population"]/housing["households"]
```

Solution: To be performed automatically in a pipeline – use base classes

```
from sklearn.base import BaseEstimator, TransformerMixin
```

Constructed class - e.g. **`CombinedAttributesAdder()`** – must support methods **`fit()`** and **`transform()`**

Transformation Pipelines

Problem: Applying transformers sequentially on feature data in the right order

Solution: Apply pipelining features from Scikit-Learn

Steps in Python – e.g.:

```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler

num_pipeline = Pipeline([
                ('imputer', SimpleImputer(strategy="median")),
                ('attribs_adder', CombinedAttributesAdder()),
                ('std_scaler', StandardScaler()),    ])

housing_num_tr = num_pipeline.fit_transform(housing_num)
```

Re-combining attributes

Problem: I may be necessary to re-combine feature attributes

Solution: Apply column transforming features from Scikit-Learn

Steps in Python – e.g.:

```
from sklearn.compose import ColumnTransformer
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
        ("num", num_pipeline, num_attribs),
        ("cat", OneHotEncoder(), cat_attribs),    ])

housing_prepared = full_pipeline.fit_transform(housing)
```

Re-combining attributes

Problem: I may be necessary to re-combine feature attributes

Solution: Apply column transforming features from Scikit-Learn

Steps in Python – e.g.:

```
from sklearn.compose import ColumnTransformer
num_attribs = list(housing_num)
cat_attribs = ["ocean_proximity"]

full_pipeline = ColumnTransformer([
         ("num", num_pipeline, num_attribs),
         ("cat", OneHotEncoder(), cat_attribs),     ])

housing_prepared = full_pipeline.fit_transform(housing)
```

Exercise

- **It is time for discussion, coding a standard regression in Jupyter**
- **Also we will investigate the housing project !!**

  - Regression Performance
  - Linear Regression Standard
  - Housing Ch. 2 No. 1
  - Housing Ch. 2 No. 2

  - ***Look at details, but don’t loose the overview ☺***
  - ***Just follow the ”right” track and you find the gold***